

# 每周工作汇报

|    |     |    |                    |
|----|-----|----|--------------------|
| 姓名 | 侯宇轩 | 日期 | 2019.7.8-2019.7.14 |
|----|-----|----|--------------------|

## 1. 本周工作

0. 时间线（标黄的为现在正在处理，标绿为已完成）

第一阶段：测试数据+CPU

1.1 准备一个浮点数测试数据。（目前小鼠数据是灰度值数据，可以直接将 90M 测试集转为浮点数先用）（发现可能不需要浮点数，直接使用整数（灰度值）的数据转为 YUV 格式）

1.2. 将其在 CPU 上转换为 YUV 4: 2: 0 格式。（目前的方法：直接修改扩展名）  
转换时通过补齐片数到 3 的倍数修正了 YUV420 格式片数不是整数片的问题

1.3. 将得到的 YUV 格式数据在 CPU 上转换为 H.264 格式。（使用他人的代码，已找到）

1.4 将得到的 H.264 格式解码为 YUV 格式。（使用他人的代码，已找到）  
decode 时出现的丢帧情况已解决

1.5 将 YUV 格式转回为浮点数测试数据。（目前的方法：直接修改扩展名）

1.6 进行比较：使用压缩解压后数据/未压缩数据造成的渲染质量的不同。

其中，1.1-1.5 需要约一周时间（6.14-6.21），1.6 需要 2 天时间（6.22,6.24）

第二阶段：测试数据+GPU

2.1 寻找 1.4 步骤的代替，设法在 GPU 上将 H.264 格式转回为 YUV 格式(Decoder)。

目前，根据任老师推荐的 Nvidia Decoder Codec SDK, 细分任务如下：

2.1.1 安装该 SDK, 运行其例子，了解其工作流程

2.1.2 编写调用 Decoder 的代码

2.1.3 使用小的真数据进行测试 Decoder（若不行，可以再换假数据测试）

2.1.4 使用更大的数据测试 Decoder，最大：4096\*4096\*4096

2.2 进行同 1.6 的测试。

2.3 将浮点数->YUV->H.264->YUV->浮点数的流程组成完整的代码。

与陶老师交流后，提出了新的要求：

目前，对 Encoder/Decoder 的调用是从文件到文件的（即：给定一个编码后的文件，输入 Decoder，输出一个解码后的文件），实际上这样的操作并不能加快体绘制的速度，因为体绘制原本要从硬盘中读取文件，现在还是要从硬盘中读取文件。

因此，陶老师希望得到的效果是，编码（压缩）后的文件通过 Decoder 后直接留在 GPU 中，用于直接从 GPU 中进行体绘制。

与此同时，数据需要分块处理，而不是整个压缩/解压，因为绘制时也是分块绘制的。

2.3.1 将数据分块（或者：压缩后进行分块排列）。

2.3.2 将封装度很高的 NVIDIA CODEC SDK 中的解码器从 文件->文件 提取出块数据->块数据的功能。

2.4 寻找 1.3 步骤的代替，设法在 GPU 上将 YUV 格式转为 H.264 格式(Encoder)。

其中，2.1 需要两周时间，(6.24-7.8)

2.2、2.3 需要一周时间 (7.9-7.15)。

2.4 需要一周时间。(7.16-7.22)

2.3 大约需要两周时间。（7.8-7.22）

第三阶段：大数据+GPU

3.1 设计如何分块处理大数据.....

2. 本周工作：

完成了 2.3.1，设置了分块方法，将测试数据分块处理。

以 128 为 block\_size，将 1780\*1368\*306 的数据分块成为

14 块\* 11 块 \* 3 块 （14\*128=1792, 11\*128=1408, 3\*128=384）

对于某个 block 数据不满  $128*128*128$  的额外的空白区域使用 0 进行填充。

由于一个 block 在 x,y,z 方向上均有（横跨 128 帧），但是原始数据存储是一帧一帧的（比如每一帧是  $1780*1368$ ，连续存储这么多字节之后才开始下一帧），因此需要不断移动文件指针来读取 block 的每一层。



这样，将其分为  $14*11*3=462$  个 block，每个大小 2048kb。

block 大小可以改变，最多不超过 1024.

## 2. 明日计划

继续 2.3.2，解读 GPU 解码的接口，试图从文件→文件转为块数据→块数据。

上周更换了显卡，现在可以正常使用 CUDA 10 了。